Team Name	**301 Programmers**	Sub-Team Name	**Mean Medians**
Sub-Team Members	Hussain Omer, Isabella Tang		
Video Presentation Link	https://drive.google.com/file/d/1ZCjWprW0EpIwpTlTY-AnX5HzNO_qNB4F/view		
PR Link	https://github.com/csc301-2025-y/project-16-rivvi/pull/42		
README Link	https://github.com/csc301-2025-y/project-16-rivvi/blob/main/README.md		

Part 1 – Planning:

Define the problem you are solving.

On our analytics page, we show graphs like employee trends and activity over time. While these are easy for sighted users to understand, they're much harder for people using screen readers. Hearing a list of numbers isn't very helpful when you can't see the overall picture.

Explain how AI can help your project (e.g., automation, decision-making, prediction, personalization, etc.).

To fix this, we use AI to generate short, clear summaries of each graph. Instead of just reading raw data, screen readers now share key takeaways like trends or big changes, making the experience much more useful.

Identify the criteria for your choice.

The approach must satisfies the following properties:

- **Dynamic**: generates responses based on the latest data
- **Fast**: delivers responses within 3 seconds after page load
- **Smart**: identifies trends and highlights key insights

Identify potential AI solutions, models, frameworks, and libraries suitable for your goal.

1. Use a pre-trained plot understanding model (e.g., PlotQA)
   These models can extract data directly from graph images and answer questions like "What's the trend over time?"
2. Apply image processing (e.g., OpenCV) to detect labels and data points, then use an LLM to generate summaries.
3. Use LLM to generate summaries directly from backend data (e.g., Gemini)
   Send structured data from the database directly to an LLM with a well-designed prompt to produce insightful summaries.
4. Train a custom model
   Build a supervised model that takes chart data as input and outputs summaries, avoiding the need for prompt engineering.

Compare alternatives and justify your choices based on feasibility (e.g., dataset availability, model complexity), performance (e.g., speed, accuracy, resource efficiency), and integration with your existing tech stack

Since the graphs are rendered in real time, cropping them and sending images to an AI model adds unnecessary complexity and delay. Because we already have access to the underlying data, analyzing it directly is both simpler and more efficient. This rules out using pre-trained visual models and computer vision libraries. Training a custom model also adds significant overhead for a task that existing language models can already handle well, so we ruled out that option as well. We chose to use a large language model to analyze the raw data directly. The data is already available from the backend, making this approach highly feasible. It's also efficient, requiring no extra preprocessing or model training. Most importantly, it fits well with our existing tech stack. We selected Gemini because it's free and fast.

If your goal involves model training/testing, outline any datasets required.

We are not training any models, so no dataset is required.

Discuss if you need to develop data preprocessing, bias mitigation and ethical.

Since the data from the database may include irrelevant information, we need to filter it to include only useful details. This ensures the AI focuses on important data, resulting in summaries that are clear, accurate, and trustworthy.

Justify why this is useful and how it aligns with the project's goals.

Our project serves a diverse range of employers, so we prioritize fairness and inclusion. This feature ensures users with disabilities are supported, making the platform more accessible and inclusive for all.

Part 2 – Implementation:

Outline the technical architecture, for example model selection (pricing, capabilities, availability, speed, etc.), data pipeline setup, API integration (if applicable)

We chose Gemini 2.0 Flash Lite for its free access and higher request per minute limits. Our backend includes API endpoints that retrieve relevant data from the database and format it into prompts tailored for AI analysis. These endpoints call a shared utility function that handles all communication with the Google Gemini API, sending prompts and receiving AI-generated summaries. On the frontend, we call these endpoints to fetch the AI responses, which are then displayed as spoken paragraphs for screen readers and as bullet-point captions for quick visual reference.

Explain, in brief, the development steps taken to use AI in your project.

We began by integrating the Google Gemini API through a utility function and building backend endpoints to handle data retrieval and prompt creation. On the frontend, we connected to these endpoints and developed components to display the AI-generated summaries. Throughout development, we focused on refining prompt design to improve the relevance and clarity of the AI's responses.

Describe any obstacles faced (e.g., computational limitations, data sparsity, deployment constraints) and how they were addressed.

During development, we encountered request limits with Gemini 2.5 Flash, which caused failures and led us to switch to the Lite version. This change required us to revise and refine our prompts to maintain consistent and accurate responses. We also faced unexpected behaviors with screen readers, which we addressed through thorough testing and learning their functionality.

Part 3 – Impact Analysis:

Compare project metric/measure before and after applying AI.

Before AI-generated summaries, the analytics page was fully visual and depended on graphs to show insights. Screen reader users had to listen to long lists of numbers without context, making it hard to understand trends. With AI, each graph now includes a real-time summary that highlights key insights like spikes, drops, and averages. Screen reader users can grasp the main idea more quickly and respond without memorizing data points. In early testing with other sub-teams, interpretation time dropped from over 20 seconds to around 5. This improvement helps not only users with accessibility needs but also anyone who wants a quick overview of the data.

Explain how the improvement was tested (e.g., unit tests, evaluations, A/B testing).

To test the improvement, we ran internal A/B testing with members from other sub-teams. They were given two versions of the analytics page, one with AI summaries and one without. We asked them to complete tasks like identifying trends (e.g., “Is employee registration increasing?”). Without summaries, users had to scan the entire graph or rely on screen readers. With summaries, they could quickly read or hear the key insights. Users completed tasks faster, made fewer errors, and felt more confident when using the AI-enhanced version. They especially appreciated the clarity around trends and averages. We also wrote unit tests to ensure clean, accurate data was sent to the AI model. These tests checked for edge cases (like empty datasets) and confirmed that API responses were successful and error-free.

Discuss trade-offs (e.g., increased latency, resource consumption, user experience improvements).

Adding AI introduced a few trade-offs. The main one was latency, since generating a summary takes about 1.5 to 2 seconds after the data loads. To minimize this, we generate the summary alongside the graph and show a loading state, so it’s usually ready when the user reaches the graph. Another trade-off was resource usage. Each AI request uses tokens, and although Gemini Flash Lite is free, it has usage limits. To stay within those, we kept prompts short and reusable. Despite these trade-offs, the user experience improved significantly. Screen reader users can now understand the graphs just as well as sighted users. The bullet-point summaries also help all users quickly understand key insights, especially when viewing multiple graphs. This feature supports our goal of making the platform more accessible.